CS361 - Group 6 - HW7: Implementation 2
Sheng Bian | Michael Waldrop | Connor Shields | Linge Ge

Software Access

This application can be directly accessed from desktop browser and mobile browser. In order to access the search and filter services functionality, the grader will need to be connected to OSU's servers, either directly or through a VPN. The rest of the website can be accessed on any network. It's also recommended to use google chrome to visit the website.

This week we have implemented further UI parts of our project and try to add database interaction for search service function. It uses bootstrap framework to optimize user experience. Users can click buttons to browse different pages. The website has a top navigation bar on each web page so users can easily find the page they want to browse.
Currently, the application has eleven pages, user can access the pages by following link:
Homepage: http://web.engr.oregonstate.edu/~bians/361/index.html
Guide List: http://web.engr.oregonstate.edu/~bians/361/guide-home.html
Guide Post: http://web.engr.oregonstate.edu/~bians/361/guide-post.html
Service List: http://web.engr.oregonstate.edu/~bians/361/service-list.html
Sample Service: http://web.engr.oregonstate.edu/~bians/361/service.html
Sign in: http://web.engr.oregonstate.edu/~bians/361/login.html
Create New Account: http://web.engr.oregonstate.edu/~bians/361/signup.html
User Profile after log in: http://web.engr.oregonstate.edu/~bians/361/profile.html
Bookmark UI Page: http://web.engr.oregonstate.edu/~bians/361/bookmark.html
Rate and Review Service: http://web.engr.oregonstate.edu/~bians/361/starRating.html
Bookmark database interaction(Need OSU VPN):http://flip1.engr.oregonstate.edu:4243/services

Finished User Story Update

Which pair(s) of teammates worked on that user story's tasks?

The schedule had some slight modifications from our earlier schedule. David and Connor were originally going to implement the bookmarking system, but we wound up re-assigning these tasks so that David and Michael worked together on setting up responsive bookmarking functionality. Michael created the html, CSS, and JavaScript, and David took this code and applied it to the framework set up by Sheng previously.

Connor and Sheng were assigned to handle the service searching, and they developed this together. This is the most important part of the website, so we created a simple database and

node application that render all services to the homepage. The services list when following the link to the services page, and there is the option to filter by category by clicking the list of categories at the top of the page. The user can also enter the type of service they are looking for, and the application will do string matching to return results which matched the string they searched for.

What do the relevant unit tests do?

The unit tests for the bookmarking page were simple. We set up a program to send queries to the database for a particular account, and if that query finds the service that it is searching for, it attaches the service to that user's bookmarked services in a foreign table that is linked to the user. That way, when the user opens their account, they can open their bookmarks, and the server will send a query for all bookmarks associated with their user id, causing the webpage to render all services the user has bookmarked. Our program then worked by sending a series of random inputs to a series of random services, and then printing those relations to the console. We then looked at the database to verify that all of the information was passed in correctly. We have not linked the database and webpages together, but we have the html setup to take user input, as well as the database. Once those are connected by proper routes from the server (which should only take 30 minutes to an hour), we will have a fully functioning site.

The tests for rendering, searching, and filtering services perform the same function. We set up a small program to send a variety of input to the database, with the query SELECT * FROM services WHERE service.name LIKE CONCAT("%", variable, "%"), where variable is the input entered by the program. The program performed small alterations on the string to send in for matching, randomly changing letters in the string to make sure the database would not return any data for incorrect information. There was no information to use to ensure the information made it to the webpage successfully. The only way we could test this was by manually querying the database, sending insertions to it and then refreshing the page to make sure it updated with all of the new information. We also manually tested the search and filter functionality, which is done by clicking on the headers above the listed services to ensure that, after clicking them, only the services under those categories are returned on the following page.

What problems, if any, did you encounter?

We did not encounter any serious problems while implementing these user stories. Developing the UI and database unfolded pretty smoothly, and the framework is set up for us to build on it more in the future, should we decide to.

How long did each task require?

Creating the database for bookmarking took one to two hours, and creating the client code for bookmarking took another three hours. We slightly underestimated the time frame, but still did not go too far over our original estimate.

What is the current status (implemented? tested?)

All of the user stories that were due by or before this week have now been implemented. The only missing functionality is the connection between the client and the database, which entails setting up the routes on the server. This aspect of the implementation should not take much more time.

What is left to be completed?

If we were to fully implement the application, we still need to connect the database to the front-end and create the user account functionality. Currently, the site operates as though there is only one user -- everyone who accesses it is basically the same user -- but we still need to set up the server so that sessions are created based on account information, which also involves more querying and setting up more tables in our database. We chose not to implement this for the user stories, because the website can be demonstrated in its full form without actually storing account information. However, we do have a client-side page that shows how the user can enter new account information and how the webpage will verify the information entered by the user.

UML Diagram/Spike Thoughts

- **Viewing Guide in Specific Category:** Back in HW 5 we submitted a UML diagram detailing the steps for this particular user story, and it was a helpful guide for determining what would be necessary for the full implementation of our project. The diagram pointed out the biggest issue with our project, which was the need for a database with which the user could interact for the specific search and filter queries. When considered against our existing setup, this presented a problem that we had to overcome. In the end, a limited database needed to be implemented in order to properly filter guides based on category. Since the full implementation of our project would require an extensive database of different types of guides, this limited database was sufficient as a proof-of-concept. For what we have, the filtering function worked correctly, and allowed the user to appropriately restrict the displayed results based on their specific needs.

- **View Top-ranked Services:** We also used a UML diagram from HW 5 for this user story. Overall, this was a simple enough implementation once the ranking system was included in the existing project framework. Clicking on Search Service simply needed to display entries in the "database" sorted in a particular way based on their ranking., which was mostly accomplished with simple redirects and a webpage set up in preparation for this particular need. Since we're working with a limited database compared to the initial vision of the customer, there are relatively few results to work with. For what we have, however, it was possible to have the services that exist on our website displayed top-down based on their star ranking.
- **Manage Bookmarks:** For bookmarks, the spike we initially presented proved to be a useful framework for a comparatively difficult-to-implement problem. Since we have been working with a limited database, our initial work was surface-level only. We were able to implement the bookmarking function using HTML and JavaScript, but this was restricted to a page-by-page basis, and did not include a centralized user account page that displayed all of the bookmarks across the entire website. Rather, the bookmark listing was restricted to the overall category of the content, such as Guides or Services. After some work, however, we were able to create a page that reflected our original intent: a user profile page listing all of the previously bookmarked items in an environment that allowed the user to manipulate them without directly accessing the content page. Had we been able to follow our initial spike to the letter, then this would probably have been easier to implement. Full implementation of the bookmark function required additional modification of our existing website mock-up, and required more work than initially planned.

Refactoring Summary

The second week saw a lot less refactoring, but we were still able to identify some areas where refactoring improved our code readability and organization.

- Moved page header into a single file. Previously the code header (Links to sign in, view guides, and view services) was literally copy and pasted on literally every single page. We realized this was a very bad practice. When the link or anything needed to be changed, we literally had to go through every single .html file that is part of our site and update it across the board. Having it as a single .html file and importing it to each ("w3-include-html" tag) html page eliminates a ton of duplicate code.
- Swapped a few id and class attributes from tags to their child html tags. Many of our.css and .js used id and class attributes located on s in order to modify to change the styling and animation of whatever content is within the the tags. We didn't quite prefer this as it obscured readability, especially when many, many tags are used in quick succession. So we removed the attributes and transferred them to

the tags' children, which allows us to read the code more easily and understand what is being modified.

- A major refactor is moving from using JS objects as a pseudo-DB to using an actual mySQL database. Website functionality stays the same, in the sense that the same data is being pulled and displayed, but this called for large swaths of code changes, since we are no longer simply requesting object property values from a .js file, and now must specifically query from a database across the network in order to retrieve and display our data.
- Separated search logic and services UI list into two separate files. Previously they were in one huge file, and the search logic was mixed within the html that was responsible for the page UI. Separating the two out in two different files allow a clear distinguishment of different functionality.
- Replacing tags with
 tags when extra spacing is desired to increase readability of html files.
- Moved a good number of inline CCS styling out of HTML tags and appended them to the appropriate .ccs file. Originally some were inline simply due to coding efficiency, but this refactor increases readability and distinguishes functionality.
- Changed services layout for better user experience and readability mainly through refactoring CCS and shuffling tags around. This is considered a refactoring as the layout is updated with change of no information or functionality.

Customer Interaction

We updated the customer with the URL of the running website, along with checking in and asking to see if the user experience matched what he envisions users would have an easy time to understand and enjoy navigating. The customer did not reply, but after working with him these past few weeks, we understand that a lack of response is a sign of approval from the customer to continue progress as-is.

The customer so far has been pleased with our current progress and liked the website system we have designed and organized. The customer has been very flexible with us, and besides working with us on the initial set of questions we asked for designing the user stories, and the subsequent decision to prioritize a subset of these stories for implementation, have been supportive of all that he has seen us working on.

There have been no surprises or any requirement changes that we encountered during the implementation process of customer's vision project. The last time such an occurrence happened was when we sought the customer to add a feature to the project in order to increase complexity and provide an additional use case for his program.

Integration Tests

After either implementing or discussing the details of implementation of the user stories for the customer, below are the results of our integration test or discussion for the system:

- Confirming that user's email is subscribed to the newsletter successfully after user submits their email at the homepage. A test email is submitted through the user interface, which gets recorded in the database, then the database is queried for that email, and a newsletter is sent to that email. The test is a success when the test email inbox receives the email sent by the system. No change is necessary.
- Confirming that all links on the website goes to the correct page. The test ensures that each URL available to all users on the website goes to the appropriate page given the appropriate permission level. The test examines any links on any of our html pages, and ensures two things: the link is not broken and the link renders the correct corresponding html page. The test results should match a hard coded URL address of the address that it should have reached. The test results caught some header links that should have led to either the guides list or services list page, but instead went to a specific service on some HTML pages. As a result, it was very prone to error to have duplicate copies of the header code in each HTML file page, and thus moved the header code to one single file that gets sourced by all appropriate pages
- Confirming that the whole create account subsystem works as expected. The user should be able to click on "Create Account," fill in the necessary information, receive a confirmation email, and account should be marked as confirmed upon verification through the confirmation email. This test was done using a preset username and test email address to create the account, and then logging into the website to make sure account is fully activated after verification. This test is one where it makes intuitive sense to develop in a test-driven approach: developing the test first before actual implementation. The implementation easily passed this test, and no changes were necessary. One additional helpful unit test that can be added would be to check that the database values are saved and match the user input. If this is omitted, the integration test can still pass account creation and verification even though the database fails to receive account creation information.
- Confirming the header link title is either "Sign In" or "Account Homepage" depending on whether the state of the browser session is logged in or not. If not logged in, the test verifies that the link to "Sign In" is populated at the header for every single page, and if the user is logged in, then the link to go to "Account Homepage" is populated at the header for every single page based on a logic check made by database query during each page rendering. The test did not pass, as the link "Sign In" is always present at the header regardless of whether the user has signed in or not. This makes sense, as we did not do dynamic login detection for client-server. Additionally, we noticed the bookmark feature was enabled even if the user was not signed in. In order for this integration test

to pass, we would need to create a flag that holds the state of whether a user is currently signed in or not. The header link and the bookmark feature would check that the flag is set in order to decide whether it should render itself or not.

- Confirming that the search finds correct data given criteria and correctly renders the articles found on the guides or services list. The test inputs a few keywords into the search box of the list pages. In the test environment, we know what the results will be when it shows up, and after it does a search, a scan is done of the returned results to ensure it matches the expected values. This is essentially a black box integration test, as the test is not concerned with the intermediate steps of what happens regarding database interaction, and simply checks the correct output given a certain input. Our search logic is pretty straightforward, and the results are correct. However, one improvement for the test would be to check more values: it wouldn't cost much more overhead, and gives us more confidence that the search logic is performing as expected.
- Confirming a submitted review matches pairs correctly with the intended service feature or guide article. Currently, the only way to write a review would be to rate the service or guide on its primary page, at which point it directs the user to the "write review" page, and returns the user back to the service/guide primary page after the review is submitted or cancelled. The test starts at a page, clicks the rate link, checks that a submitted to the associated service/guide and that a cancelled review makes no changes in the database. Our website fails this test, due to the fact that we were not able to fully implement the backend review guide/service functionality. User is able to go to the review page from the guide/service page, but upon submitting the review, nothing is recorded. In order to pass this test, we have to add the implementation to associate any review with a service/guide in a many-to-1 relationship.
- This is not an integration test we wrote, simply because in the scope of this class we would not be able to implement this, but one complex subsystem is the ability to allow Security Service Providers to host their service feature page and allow users to pay/download their service. Something like this would definitely require several integration tests, and an essential test we discussed is to validate the correct relationships between payment and downloading. This test would check that downloading is not possible if a user has not paid, and once a user has paid, it checks for verification of payment and that downloading is a possibility. If this integration test fails between payment and downloading, then it is essential that a fix becomes readily available to block downloading until confirmed payment is detected correctly.

Schedule For Next Week

We worked this week with the goal of finishing implementation for all user stories customer assigned. Unfortunately, our workload consisted of many other "projects" we had to balance for other "customers" and we did not meet customer's requirement that all items below be

finished. Hence, if we had a next week, this an updated, prioritized list of what we would work on and how long we would spend doing it:

- Viewing Guide In Specific Category (**Due Date:** ***Completed***)
    - **Tasks:** ~~Develop a homepage UI, and a clickable button leading to the guides page UI. From here, users can see a list of different guides in a populated list, with tabs at the top for category, date uploaded, rating, etc. Clicking on category will sort the guides by content type. Clicking on a guide name will list the title, author, date written, and a text block with information about the topic. At the bottom of the guide page, a small list will be populated with the most popular services for that content type.~~
    - **Est. Implementation Time:** 10 → **4** → ***0 Done***
    - **Programmer(s):** Sheng, Connor
- View top-ranked Services (**Due Date: 6/12**)
    - **Tasks:** ~~Develop homepage UI and clickable services button. Clicking the services button will present a populated list of services sorted by rating.~~ The top of the services list will have several clickable filter options, such as category for sorting services by specific technology types.
    - **Est. Implementation Time:** 9 → **3** Pair-Programming Hours
    - **Programmer(s):** Michael, Sheng
- Manage Bookmarks (**Due Date: 6/12**)
    - **Tasks:** Develop a user account homepage with a list of previously bookmarked guides and services based on the user's account identity. This page will have clickable delete buttons for the bookmarks, which will allow users to remove the bookmark form their user account database of bookmarked items. ~~Develop a clickable bookmark button for each guide and service entry in the primary database that will link the viewed item to the user account identity, and copy/link the viewed item to the user's bookmark database.~~
    - **Est. Implementation Time:** 4 → **3** Pair-Programming Hours
    - **Programmer(s):** David, Connor
- Rate and Review Service (**Due Date:** ***Completed***)
    - **Nice-to-Have:** ~~connecting our rate/review service form to each individual service page.~~
    - **Est. Implementation Time:** 2 → **1** → ***0 Done***
    - **Programmer(s):** Michael, David
- Parents Protecting Their Kids (**Due Date:** ***Completed***)
    - **Est. Implementation Time:** -
    - **Programmer(s):** Michael, David
- Create account and subscribe to newsletter for updates (**Due Date:** ***Completed***)

- **Est. Implementation Time:** -
- **Programmer(s):** Sheng, Connor

Customer Contribution

The customer reached out to us requesting the URL of the project, and we responded within 24 hours.

Team Contribution Summary

Sheng: Code the UI pages of user profile, bookmark UI page and service list. Integrate team members' code work into the main repo. Wrote software access part.
David: Refactor summary, integration tests overview, next week schedule, customer interaction update. Implemented integration of several features: rating to individual services, response to newsletter sign-up, search services on homepage, service list page (before being refactored).
Connor: Implemented the database and server to host the web pages for filtering, searching, and displaying services. Wrote the finished user story update.
Michael: Implemented the bookmark function on the guides page, wrote the UML/Spike thoughts for this week.